CHUYÊN ĐỀ
TỔ CHỨC DỮ LIỆU
CÁC KIỂU DỮ LIỆU NÂNG CAO 3
LỚP

Giảng viên: VŨ QUỐC HOÀNG
(vqhoang@fit.hcmus.edu.vn)

Nội dung trình bày

Kiểu lớp (Class)

- (Nhắc lại) Kiểu cấu trúc là kiểu dữ liệu gồm nhiều thành phần dữ liệu:
  - Các thành phần có thể khác kiểu nhau
- Kiểu lớp là kiểu dữ liệu gồm nhiều thành phần dữ liệu:
  - Các thành phần có thể khác kiểu nhau
    - Các thành phần có thể có kiểu là kiểu hàm
- (Nhắc lại) Kiểu hàm là kiểu cho phép quản lý, thao tác trên các hàm:
  - Hàm cũng là dữ liệu
  - Kiểu hàm được hiện thực bằng con trỏ hàm (C/C++)

Kiểu lớp
Thuật ngữ

- Class = Fields + Methods
- Object / Instance / Class = Value / Data / Type
- Object = State + Behavior
  - Class: Lớp
  - Field: Trường
  - Method: Phương thức
  - Object: Đối tượng
  - Instance: Thể hiện
  - State: Trạng thái
  - Behavior: Hành vi

Kiểu lớp
Ví dụ mã máy / hợp ngữ

Kiểu lớp
Ví dụ C


```
struct Point
{
    int x, y;
    void (*draw)(struct Point*);
};
void f(struct Point *self)
{
    printf("(%d, %d)", self->x, self->y);
}
void main()
{
    struct Point *p = (struct Point*)malloc(sizeof(struct Point));
    p->x = 10; p->y = 20;
    p->draw = f;
    p->draw(p);
}
```

Con trỏ là quan trọng

Kiểu lớp
Ví dụ C++


```
struct/class Point
{
public:
    int x, y;
    void draw();
};
void Point::draw()
{
    printf("(%d, %d)", this->x, y);
}
void main()
{
    Point *p = new Point;
    p->x = 10; p->y = 20;
    p->draw();
}
```

Có sự hỗ trợ của Trình biên dịch

Kiểu lớp
Ví dụ C#


```
class Point
{
    public int x, y;
    public void draw()
    {
        Console.Write("({0}, {1})", this.x, y);
    }
}
class Program
{
    static void Main()
    {
        Point p = new Point();
        p.x = 10; p.y = 20;
        p.draw();
    }
}
```

-Tham chiếu là tự nhiên
-Phân biệt rõ trường với phương thức

Tiến hóa kiểu

- Các kiểu dữ liệu tiến hóa theo hướng: Máy ➔ Người
- Kiểu lớp rất gần gũi với người nhưng vẫn có thể hiện thực trên máy
- Các nhân tố ảnh hưởng đến sự tiến hóa kiểu
  - Nhân cách hóa
  - Trừu tượng hóa
  - Tiện lợi hóa
  - Tự vệ
  - Kế thừa
  - Đa hình
  - Tự khám phá
  - …

Nhân cách hóa

- Làm cho dữ liệu sống động như người (hoặc ít nhất, như là một sinh vật):
  - có hành vi
- Vòng đời (life cycle) của một đối tượng:
  - Sinh: Constructor
  - Hoạt: Behavior
  - Tác: Messaging
  - Tử: Destructor

Nhân cách hóa


```
class Human
{
    private int age;
    public int Age
    {
        get { return age; }
    }

    public Human()
    {
        Console.Write("Oe!");
        age = 0;
    }

    public void An_Choi_Ngu()
    {
        age++;
        Console.WriteLine(age);
    }

    ~Human()
    {
        Console.Write("Bye!");
    }
}
```

```
class Program
{
    static void Main()
    {
        Human ahuman = new Human();
        while (ahuman.Age < 90)
            ahuman.An_Choi_Ngu();
    }
}
```

Constructor

Behavior

Destructor

Messaging

Trừu tượng hóa (Abstraction)

- Trừu tượng hóa là khả năng quan sát vật/việc mà không cần xem xét đến các chi tiết bên trong nó:
  - Trừu tượng hóa chức năng: không để ý “làm thế nào?”, chỉ cần biết “làm được gì?”
  - Trừu tượng hóa dữ liệu: không cần để ý đến chi tiết, định dạng cụ thể các thành phần
- Kiểu lớp có mức trừu tượng vừa phải:
  - Không quá đến mức nằm “trên giấy” như các cấu trúc dữ liệu trừu tượng
  - Có thể hiện thực trên máy như các kiểu dữ liệu cụ thể khác
  - Gần gũi với con người

Trừu tượng hóa


```
interface Stack
{
    void Push(int i);
    int Pop();
    bool IsEmpty { get; }
}
class RealStack : Stack
{
    // ... (fields)
    // ... (methods)
}
```

```
class Program
{
    static void Main()
    {
        Stack s = new RealStack();
        int n;
        while (n != 0)
        {
            s.Push(n % 2);
            n = n / 2;
        }
        while (!s.IsEmpty)
        {
            Console.Write(s.Pop());
        }
    }
}
```

Tiện lợi hóa

- Tiện lợi hóa là nhu cầu tích hợp chức năng vào sản phẩm
  - Hiện thực bằng sự đóng gói (encapsulation):
    - không tách bạch dữ liệu và chức năng (thao tác)
    - mà, kết hợp dữ liệu với chức năng vào một đơn vị duy nhất

Tiện lợi hóa


```
struct XXX
{
    int a, b;
};
void work1(XXX &x) { ... }
void work2(XXX &x) { ... }
...
void work10(XXX &x) { ... }
Void main()
{
    XXX x = {1, 1};
    work1(x);
    work2(x);
    ...
    work10(x);
    printf("%d, %d", x.a, x.b);
}
```

```
class XXX
{
    public int a, b;
    public void work10in1()
    {
        ...
    }
}
class Program
{
    static void Main()
    {
        XXX x = new XXX();
        x.a = x.b = 1;
        x.work10in1();
        Console.Write("{0}, {1}",
                          x.a, x.b);
    }
}
```

Tự vệ

- Tự bảo vệ là khả năng che chắn trạng thái không cho thế giới bên ngoài trực tiếp thay đổi:
  - Mang lại tính riêng tư
  - Chỉ cho phép thay đổi trạng thái (gián tiếp, dưới sự kiểm soát của chính đối tượng) thông qua hành vi
  - Hiện thực bằng:
    - Bổ từ truy cập (Access modifier)
    - Thuộc tính (Property)

Tự vệ


```
class Human
{
    private int money;
    private int age;
    public int Age
    {
        get { return age; }
    }

    public void An_Choi_Ngu()
    {
        age++;
    }

    public void Work()
    {
        age++;
        money++;
    }
}
```

Bổ từ truy cập

Thuộc tính

```
class Program
{
    static void Main()
    {
        Human ahuman = new Human();
        ahuman.age = 100; // error
        ahuman.An_Choi_Ngu();
        ahuman.Work();
        Console.Write(ahuman.Age); //ok
        Console.Write(ahuman.money); //error
        Console.Write(ahuman.Money); //error
    }
}
```

Kế thừa (Inheritance)

- Kế thừa là khả năng một đối tượng thừa hưởng các đặc tính của cha mẹ:
  - Mang lại khả năng tái sử dụng
  - Kế thừa trong các ngôn ngữ là kế thừa ở mức lớp: một lớp con có thể dùng lại, mở rộng hoặc thay đổi các thành phần của lớp cha
    - Lớp cơ sở: base class/super class
    - Lớp kế thừa: derived class/sub class

Kế thừa


```
class Man
{
    protected Clothes clothes;
    public void eat()
    {
        Console.Write("ngoam, ngoam");
    }
    virtual public void move()
    {
        Console.Write("go, go");
    }
}

class SuperMan : Man
{
    private Mask amask;
    override public void move()
    {
        Console.Write("fly, fly");
    }
}
```

```
class Program
{
    static void Main()
    {
        SuperMan spiderman = new
                                SuperMan();
        spiderman.eat();
        spiderman.move();
    }
}
```

Đa hình (Polymorphism)

- Đa hình là khả năng thay đổi hình thái trong vòng đời của đối tượng
  - Trong các ngôn ngữ thì, thực tế, đó là khả năng thay đổi hành vi trong vòng đời của đối tượng với cùng một tác động của một trường
    - Hiện thực hóa bằng kết buộc động (dynamic binding): khả năng lựa chọn phương thức thực thi khi đối tượng nhận thông điệp vào lúc chạy

Đa hình


```
abstract class Animal
{
    abstract public void speak();
}
class Cat : Animal
{
    override public void speak()
    {
        Console.Write("Meo, Meo");
    }
}
class Dog : Animal
{
    override public void speak()
    {
        Console.Write("Gau, Gau");
    }
}
class Cow : Animal
{
    override public void speak()
    {
        Console.Write("Umm ... bo");
    }
}
```

```
class Program
{
    static void Main()
    {
        Animal[] pets = new Animal[3];
        pets[0] = new Cat();
        pets[1] = new Dog();
        pets[2] = new Cow();
        foreach (Animal pet in pets)
            pet.speak();
    }
}
```

Tự khám phá

- Tự khám phá là khả năng tự nhận biết chính mình
  - Tự nhận biết mình là ai? (nhận biết kiểu): đa hình
  - Khi thực hiện một hành vi, đối tượng có khả năng nhận biết trạng thái hiện tại của mình và có khả năng thực hiện các hành vi khác của mình
    - Hiện thực bằng tham chiếu `this`: tham chiếu đến bản thân đối tượng đang thực hiện hành vi

Tự khám phá


```
class Car
{
    int speed;
    public void Start()
    {
        speed = 1;
    }
    public void SpeedUp()
    {
        if (this.speed == 0)
            Start();
        else if (speed < 125)
            speed++;
    }
}
```

Hỏi và Đáp